Los Contenedores

AISLAMIENTO, SÍ, PERO... ¿DISTRIBUCIÓN?

Por Gunnar Wolf

Gunnar Wolf es administrador de sistemas para el Instituto de Investigaciones Económicas de la UNAM y desarrollador del proyecto Debian GNU/Linux.
http://gwolf.org

Discutir acerca de mecanismos de virtualización implica cubrir una gran cantidad de tecnologías distintas. Y no me refiero con esto a diferentes proveedores que ofrecen productos con funcionalidad similar, sino que a herramientas de muy distinta naturaleza, que a veces incluso no parecen tener nada que ver entre sí.

La primera acepción que vendrá a la mente de muchos, es la virtualización asistida por hardware, donde una computadora con ciertas capacidades de hardware (disponibles en la arquitectura x86 desde hace ya diez años, y hoy presentes incluso en los CPUs de más baja gama) ejecuta un hipervisor, software específico que se ubica por debajo del sistema operativo y permite que la computadora sea compartida entre distintos sistemas operativos (o distintas instancias del mismo), aparentando ante cada uno de ellos ser una computadora independiente. Como apéndice de esta categoría entrarían los paravirtualizadores, que corren versiones de dichos sistemas operativos que saben que se ejecutarán dentro de una máquina virtual, con lo cual delegan algunos procesos al sistema anfitrión, muchas veces aumentando la estabilidad o reduciendo la penalización de velocidad.

Por otro lado, la emulación de hardware completamente distinto del sistema primario puede también considerarse virtualización. Podemos encontrarla como parte de los kits de desarrollo para dispositivos móviles, en que el código generado se compila para procesadores típicamente de arquitectura ARM, mientras que el desarrollo se efectúa sobre computadoras x86. La emulación incluso cubre a las máquinas virtuales empleadas nativamente, como la JVM (Java) o CIL (.NET): El bytecode compilado para estas arquitecturas no corre nativamente en ningún procesador, son arquitecturas diseñadas para ser siempre emuladas.

Una modalidad más de virtualización es el uso de contenedores. Esta modalidad puede comprenderse mejor si se contrasta con las anteriores dónde está el engaño — O (más bonito) dónde está la magia. En vez de proveer una computadora virtual a los sistemas operativos huéspedes, los contenedores buscan proveer un sistema operativo virtual a conjuntos de programas.

¿Qué significa esto? Que a cambio de reducir la flexibilidad, esta modalidad de virtualización provee un mucho mejor rendimiento (las aplicaciones virtualizadas tienen exactamente la misma velocidad que en el hardware nativo) y menor impacto en el resto del sistema (cuando los procesos que forman parte del contenedor no tienen trabajo por realizar, su consumo de recursos es verdaderamente cero, a diferencia de un sistema virtualizado, que debe seguir pendiente a posibles eventos).

La reducción de flexibilidad referida consiste en que, dado que el núcleo del sistema operativo en ejecución es uno solo, todos los procesos que sean ejecutados empleando esta técnica deben estar compilados para la misma arquitectura y sistema operativo —esto significa que por ejemplo, en un sistema anfitrión x86 con Linux, todos los contenedores deberán ser también x86 con Linux. Esta tecnología hereda directamente de la llamada al sistema chroot(), en 1982. Claro está, a esta simple llamada se tuvieron que agregar numerosas funciones implementando una separación más limpia y completa entre los distintos espacios de nombres. Poco a poco se desarrollaron mecanismos para limitar el uso total de memoria o de CPU de cada contenedor, y aislar la vista de la red que cada uno de ellos puede tener. El primer sistema en implementar lo que hoy conocemos con contenedores fue FreeBSD, en el año 2000; dado que están construidos alrededor de la llamada chroot(), resulta natural que casi todos los sistemas operativos en implementar contenedores sean tipo Unix; en el mundo Windows, hoy en día se tiene la promesa de que la versión 10 incluirá esta característica.

En Linux hay distintas implementaciones disponibles, y que han logrado madurar cada cual a su ritmo. Durante muchos años, la más fuerte fue VServer, pero requería de cambios demasiado invasivos, que nunca fueron aceptados por Linus Torvalds para la rama principal del kernel. Posteriormente se popularizó OpenVZ, la versión libre de Virtuozzo, pero sufrió del mismo problema. Sin embargo, a partir de la incorporación de los grupos de contexto (cgroups) al kernel de Linux 2007, el proyecto Linux Containers (lxc) logró una implementación suficientemente eficaz y sencilla, y es hoy en día la más popular y consolidada.

Distribución basada en contenedores

A principios de 2013, se anunció la liberación de un innovador programa que ha cambiado en gran medida las reglas del juego para el despliegue de aplicaciones: Docker. Su principal contribución es que cambia

> "Docker nos invita a olvidarnos de la seguridad que debe mantenerse como parte de la gestión de cada sistema instalado".

el enfoque: presenta a cada contenedor ya no como una máquina virtual, como una instalación completa de una computadora, sino que como el entorno completo de ejecución de una aplicación. Esto tiene aspectos muy positivos, pero también tiene otros francamente dignos de cuidado. Pero antes de abordarlos, permítanme explicar un poco.

Originalmente, instalar o aprovisionar un contenedor implicaba como paso necesario la instalación del software de sistema tal como si se tratara de una computadora real. La delgada capa de virtualización de los contenedores permitían tomar a un conjunto de máquinas virtuales como si fuera una granja de servidores — Y el trabajo del administrador de sistemas es, a fin de cuentas, gestionar a cada una de las máquinas virtuales cual si fuera un servidor completo — Que si bien es mucho menos complejo que si se tratara de una sola instalación con todo el software bajo el mismo espacio, implica una innegable complejidad. El principal cambio que introduce Docker es que permite generar un contenedor por aplicación, y gestionarlo directamente como si fuera una unidad — ya no una colección de paquetes relacionados.

Bibliotecas, versiones, infiernos y apps

Demos un brinco conceptual a un tema, aparentemente, sin relación. Prometo pronto explicar a qué viene esto.

El desarrollo de los sistemas operativos que empleamos hoy en día cruzó hace unos treinta años por un punto crucial en lo concerniente a su flexibilidad: La introducción de formatos estándar de bibliotecas compartidas, que pueden ser utilizadas entre diversas aplicaciones. Gracias a ellas, el tamaño de cada uno de los programas se redujo fuertemente (puesto que éstos no tenían que incluir expresamente todas las definiciones que emplean), se estandarizaron interfaces (puesto que diversos programas aprovecharían las mismas bibliotecas ampliamente conocidas), y –muy importante– dado que redujo fuertemente la duplicación de código, mejoró significativamente la seguridad de los sistemas operativos: Ante un error o vulnerabilidad, basta con actualizar una copia del código vulnerable para que todos los programas afectados reciban la corrección.

Claro, surge complejidad del manejo de los cientos de bibliotecas compartidas que se instalan en un sistema típico, de la cual se deriva la expresión "DLL hell" (el infierno de los DLLs). Sin embargo, este es un problema en el que se ha invertido una gran cantidad de trabajo, y la situación hoy en día es radicalmente diferente a la de hace veinte años, cuando se acuñó dicho término.

Ahora bien, ante la popularización de las apps en dispositivos móviles y el abaratamiento del almacenamiento, se impuso el concepto inverso: El empaquetamiento de todas las dependencias dentro de un mismo binario. Para evitar que el usuario final tenga que descargar todas las dependencias que empleó cada desarrollador, y asegurar que cada paquete funcionará independientemente de los demás que tenga instalado el sistema, se ha optado por –de facto– reconvertir las bibliotecas dinámicas en objetos ligados estáticamente. Claro, esto obliga a que ante la actualización de una biblioteca compartida, cada una de las aplicaciones que la usa sea actualizada también — Pero en una era de conexión permanente a Internet, esta idea ha resultado del favor del mercado.

Distribución y servidores virtuales

Uniendo, pues, los dos temas expuestos: La principal contribución de Docker es que convierte el engorroso despliegue de servidores virtuales en un sencillo despliegue de aplicaciones — Y, claro está, de aplicaciones empaquetadas a la moda, con el mínimo de dependencias.

Desde una perspectiva meramente de DevOps, esto suena muy bien. Sin embargo, también involucra grandes riesgos, como lo ilustran entre otros muchos los textos que han escrito al respecto dos de los desarrolladores de Debian GNU/Linux: Joey Hess [1] y Erich Schubert [2].

En primer lugar, un uso irresponsable de Docker (aunque sea el recomendado por sus desarrolladores) nos puede llevar a confiar en software cuyo origen no necesariamente es el que esperamos, sin saber en qué consisten los cambios y cómo éstos nos afectarán. Los paquetes que dicen ser oficiales de determinadas distribuciones han demostrado haber sido alterados, lo cual debe poner a pensar mal a cualquier administrador de sistemas.

En segundo lugar, y esto me parece más complicado y peligroso aún, el modelo que impulsa Docker nos invita a olvidarnos de la seguridad que debe mantenerse como parte de la gestión de cada sistema instalado. Un contenedor no puede ser creado una única vez y visto, a partir de ese momento, como una app, como un objeto estático e independiente. Y no, cada sistema debe integrar las correcciones a las vulnerabilidades (o simples bugs) que constantemente van apareciendo y siendo corregidas, y un administrador de sistemas siempre debe estar familiarizado con la tecnología que tiene instalada. Resulta inexcusable poner al mismo nivel a un sistema operativo completo y a aplicaciones como Wordpress, pero eso es precisamente lo que ocurre [3].

Obviamente, la tecnología no es mala, y la adopción que ha tenido habla de sus virtudes técnicas. Sin embargo, el uso y crecimiento que ha tenido dejando de lado las buenas prácticas de administración de sistemas, y el impacto que esto puede tener en nuestro campo, merecen ser consideradas.

Referencias

[1] "What does docker.io run -it debian sh run?", https://joeyh.name/blog/entry/docker_run_debian/

[2] "The sad state of sysadmin in the age of containers", http://www.vitavonni.de/blog/201503/2015031201-the-sad-state-of-sysadmin-in-the-age-of-containers.html

[3] Repositorios oficiales de Docker: https://registry.hub.docker.com/search?q=library&f=official